Linux상에서 확장 가능한 VOD시스템의 설계 및 구현

김정원[†] · 김인환[††] · 정기동[†††]

요 약

주문형 비디오 시스템은 향후 도래할 멀티미디어 서비스 시대의 핵심 응용 분야이다. 본 연구에서는 최근 엔터프라이즈나 연구용으로 각광받고 있는 Linux 커널상에서 저가형 PC 클러스터링 환경을 구축하여 호스트 단위로 확장 가능한 VOD시스템(SVOD : Scalable VOD)을 설계 및 구현하였다. 본 연구의 주요 기여는 다음과 같다. 첫째, MPEG과 같은 연속매체를 서비스하기 위해서 기존의 텍스트용 Linux Ext2 파일시스템을 디스크 배열상에서 VOD전용으로 개발하였다. 둘째, 호스트 단위로 확장가능하도록 스트림 서버를 구축하였다. 셋째, 클라이언트는 마이크로소프트 DirectShow® COM(Component Object Model)을 이용하여 하드웨어 지원없이 소프트웨어 디코더를 구현하였다. 넷째, 서버와 클라이언트 사이의 흐름제어를 이용하여 클라이언트 버퍼의 Overflow와 Underflow 발생을 억제하고, 이를 통한 FF(Fast Forward) VCR연산을 제공한다. 연구결과, 저가의 PC서버와 무료 운영체제인 Linux상에서 쓰레드 기반의 확장가능한 VOD시스템 개발이 가능하였다.

Design and Implementation of Scalable VOD System on Linux

Jeong-Won Kim[†], In-Hwan Kim[††] and Ki-Dong Chung[†††]

ABSTRACT

Video on Demand (VOD) system is definitely one of main applications in upcoming multimedia era. In this research, we have designed and implemented a host-based scalable VOD system (SVOD) which is composed of low cost PC servers and runs on Linux kernel that is currently spotlighted in enterprise and research domains. Our contribution is as follows: first, the previous Ext2 file system was modified to efficiently support continuous media like MPEG stream. Second, the storage server features a host-based scalable architecture. Third, a software MPEG decoder was implemented using Microsoft's DirectShow® COM. Finally, flow control between client and server is provided to suppress overflow and underflow of client circular buffer and supports FF VCR operation. We have known that it is possible to develop a thread-based and scalable VOD system on low cost PC servers and free Linux kernel.

1. 서 론

컴퓨터, 저장 장치 및 통신 기술의 발전은 동영상이나 음성을 실시간으로 서비스할 수 있게 되었다. 특히 주문형 비디오 서비스의 단말 사용자는 영화, 뉴스, 쇼핑 등의 디지털 동영상 데이터를 실시간으로 각 가정에서 선택할 수 있다. 이것은 이전까지의 영상 매체들이 서비스 제공자가 제공하는 스케줄에 의해 동시에 일괄적으로 수용하는 구조를 탈피하여 사용자가 원하는 컨텐트를 원하는 시간에 얻을 수 있음을 의미한다. 또한 사용자는 VCR 뿐만 아니라 서비스의 질(QoS : Quality of Service)도 조절할 수 있게 되었다.

한편, 개인용 컴퓨터는 450MHz의 클럭 스피드로 MPEG과 같은 압축된 동영상을 하드웨어의 지원 없이 실시간으로 플레이백할 수 있다. 하드디스크 드라이브는 10GB용량의 데이터를 저장할 수 있으며[1], 60분용 MPEG II 비디오가 2 기가바이트 정도 차지

본 연구는 1999년도 대학기초사업 연구비에 의해 연구되었음.
† 부산대학교 전자계산학과 대학원 박사수료
†† 부산대학교 전자계산학과 대학원 박사과정
††† 부산대학교 정보 · 컴퓨터 공학부 교수

하므로 5편 이상을 저장할 수 있다. 또한 개인용 컴퓨터나 워크스테이션의 입출력 인터페이스로 주종을 이루는 SCSI 버스의 경우 이론적으로 초당 160 메가바이트의 대역폭을 지원하고[2], 평균 대역폭 12.4MB/sec 인 디스크들이 버스 대역폭을 공정하게 사용한다면 이상적인 경우 12~15개까지 장착할 수 있다. 또한 기간망으로 구축되고 있는 ATM(Asynchronous Transfer Mode), ADSL(Asymmetric Digital Subscriber Line), Gigabit ethernet, FastEthernet 등은 연속매체를 전송하기에 충분한 대역폭을 지원하고 있다. 따라서, 기존의 연구들이 고가의 서버로 시스템을 구축하였는데, 이것은 시스템 확장시 비용적인 측면에 부담이 될 수 있다. 위의 제반 사항들을 감안해 보면 저가의 PC서버상에서도 연속매체 사용자를 충분히 지원할 수 있다.

또한, 연속매체를 지원할 경우 동시 사용자수를 증가시키기 위해서는 멀티미디어 파일시스템, 실시간 디스크 스케줄링, 실시간 쓰레드 스케줄링 등의 커널 수준의 지원이 필요하다. 이에 대한 연구들은 이미 상당히 진행된 상태이다[3,4,5]. 커널 개발자의 입장에서 기존의 상용 운영체제는 라이센스 비용의 부담과 개발 기간의 장기화 등의 문제가 있다. 한편, Unix와 호환이 되는 Linux는 현재 학계나 기업에서 각광 받고 있는 무료 운영 체제이다. 이미 오라클이나 인포믹스와 같은 대형 DBMS 사업자들이 리눅스 지원을 하기 시작했으며, 대학에서도 활발이 연구중이다. 이것은 리눅스가 이미 안정된 운영체제로 엔터프라이즈 시장에서도 이미 그 가능성을 인정받았음을 의미한다.

본 연구에서는 Linux커널과 저가형 PC서버상에서 확장가능한 소규모 VOD시스템을 설계 및 구현하였다. 연구의 주요 내용은 다음과 같다. 연속매체를 커널의 파일시스템 차원에서 효율적으로 지원하기 위해서 Linux Ext2 파일시스템을 개량하여 미디어 파일시스템(MFS : Media file system) 을 설계 및 일부 구현하였다. 또한, PC 클러스터상에서 소규모로 확장 가능하도록 VOD서버 프로그램을 구조화하였다. 기존의 VOD시스템의 클라이언트는 하드웨어 디코더 사용이 일반적인데 본 연구에서는 마이크로소프트사의 다매체 디코더인 DirectShow® COM 인터페이스를 이용하여 소프트웨어 디코더를 구현하였다. 또한, 클라이언트와 서버 사이의 흐름제어를 통하여 jitter 발생을 억제하고 Fast forward VCR 연산을 지원한다.

논문의 구성은 다음과 같다. 2장에서 관련연구를 살펴보고, 3장에서는 SVOD 시스템 개괄적 구조를 설명하고, POSIX 쓰레드 기반 SVOD 서버 구조를 제시한다. 4장에서는 DirectShow를 이용한 클라이언트 구조를 설명하고 5장에서 서버와 클라이언트의 통신 프로토콜과 흐름제어 방법을 제시하고, 6장에서는 구축된 시스템의 성능 평가 결과를 설명한다. 마지막으로 7장에서는 결론과 향후 연구 과제를 기술한다.

2. 관련연구

다음은 관련 연구로서 기존의 VOD시스템을 설명한다. Fellini 멀티미디어 저장 서버[6]는 AT&T Bell Labs에서 개발한 것으로 독립적인 중저가형 미디어 서버를 기반한 클러스터링 형태의 서버이다. 이 서버는 멀티미디어 응용과 범용 응용을 동시에 지원하기 위해 실시간 데이터에 대한 저장 및 검색 기능을 제공한다. Southern California 대학에서 개발한 Mitra 연속 미디어 서버[7]는 클러스터 기반의 연속 미디어 서버이다. Mitra는 계층적 저장 장치를 채택한 저장 서버를 중심으로 확장성을 제공하고, 다양한 종류의 미디어에 대한 저장 및 검색 기능과 비디오 스트림에 대한 VCR 연산을 지원한다. France Eurocom Research Center에서 개발한 Video server array[8]는 클러스터 기반의 연속 미디어 서버이다. 분산 수평 구조로 시스템이 구성되며, 다수의 비디오 서버가 분산 파일 시스템을 통하여 미디어 데이터를 공유하고 병렬 입출력을 지원한다. Microsoft의 Tiger Video Fileserver[9]는 분산구조이고 고장 감내 실시간 파일 서버로서 동종의 컴퓨터와 디스크들이 고속 네트워크에 결합된 구조이다. 데이터는 소속된 전 컴퓨터 및 디스크에 스트라이핑 되며, 정해진 스케줄에 의해 라운드 로빈 기법으로 읽혀진다. Starlight의 StarWork^{TM} [10]는 다양한 형태의 비디오 서비스를 제공하는 디지털 비디오 네트워킹 소프트웨어이다. 주요 관심은 하드웨어 RAID가 아닌 소프트웨어 스트라이핑을 사용한다는 점이다.

(a) 시스템 환경

(b) 시스템의 논리적 연결

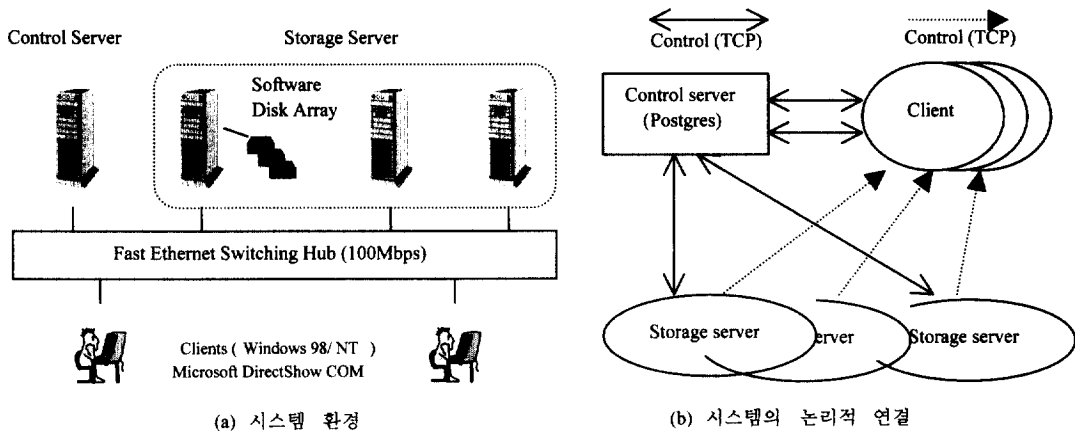



그림 1. SVOD 시스템의 구조

3. SVOD 시스템 구조

3.1 시스템의 구조

소규모의 비디오 서버는 관련 연구의 고가 시스템으로 구축하기 보다는 그림 1과 같이 저가의 서버를 클러스터링하여 병렬화 시키는 것이 비용적인 측면에서 유리하다. SVOD는 소규모 환경에 적합하도록 저가의 PC서버를 사용하여 시스템을 구축하고자 한다. 그림 1은 구축된 SVOD 시스템의 전반적인 환경과 시스템 구성요소 간의 논리적인 연결 구조이다. 그림 1의 (a)는 전반적인 구조인데 하나의 제어 서버(Control server), 다수의 스토리지 서버(Storage server), 그리고 이들을 상호 연결하는 고속의 내부 네트워크로 구성된다. SVOD의 확장성은 스토리지 서버의 수에 좌우되는데 다음의 예를 고려하자. 예를 들어, 한 스토리지 서버에 대역폭이 40Mbytes/sec인 SCSI 버스가 세 개 장착되었고, 디스크는 평균 대역폭이 12.4 Mbytes/sec, 비용이 α이고, 대역폭이 1.5Mbps인 MPEG-I 스트림을 서비스한다고 가정하자. Storage server A는 한 버스에 3개의 디스크를 장착하고, Storage server B는 6개를 장착할 때, A는 9α의 저장 장치 비용으로 버스 대역폭을 공정하게 사용한다면 이론적으로 최대한 595.2개(12.4Mbytes/sec * 9 Disks / 1.5Mbits/sec)의 스트림을 지원할 수 있다. 반면, 시스템 B는 18α의 비용으로 최대한 640개(40Mbytes/sec * 3 Bus / 1.5Mbits/sec)의 스트림을 지원할 수 있다. 이것은 한 버스에 소속된 디스크의 총 대역폭이 버스의 대역폭을 초과하기 때문이다(12.4Mbytes/sec * 18 Disks > 40Mbytes/sec * 3 Bus). 시스템 B는 비디오 저장은 A보다 2배 가량 저장할 수 있으나, 두 배의 저장 장치 비용으로 두 배의 스트림을 지원할 수 없다. 한편, 시스템 A와 같이 버스 대역폭만 고려하여 디스크를 장착하는 것은 스토리지 서버의 수만 증가 시킬 수 있다. 따라서, 시스템 관리자는 시스템 구축 총 비용과 지원 사용자 수의 Trade-off를 고려하여 결정해야 한다.

그림 1의 (b)는 서버와 클라이언트와의 논리적인 연결 구조를 표현한다. 확장성을 고려하여 제어 서버가 클라이언트의 모든 요구를 처리하고, 스토리지 서버는 클라이언트에게 투명하며 연속적인 데이터 전송을 책임진다. 즉, 제어 서버는 클라이언트가 비디오를 요구할 때 비디오를 저장하고 있는 스토리지 서버로 데이터를 전송할 것을 명령한다. 이것은 스토리지 서버의 동적인 추가에도 적용할 수 있는 구조이고, SVOD 전체의 부하 균형을 획득할 수 있다. 제어 서버와 스토리지 서버, 제어 서버와 클라이언트 간의 메시지는 TCP/IP로 신뢰성을 보장하고, 스토리지 서버는 클라이언트와 UDP/IP로 데이터 패킷을 전송하게 된다. 이것은 제어와 데이터 메시지를 분리하여 원활한 전송을 보장할 수 있다. 데이터 패킷으로 UDP를 사용한 것은 비연결 구조인 UDP가 속도가 TCP보다 월등하므로 지원 가능 스트림의 수가 증가되고, 제안된 SVOD가 소규모 응용이므로 연속매체를 서비스할 정도의 신뢰성은 얻을 수 있기 때문이다.

3.2 POSIX 쓰레드 기반 서버 소프트웨어 구성

Linux는 POSIX 쓰레드를 지원하며, 쓰레드는 프로세스와 동등하게 스케줄링된다[11]. VOD 응용은 실시간에 연속성을 보장해야 하므로 동시성과 병행성을 갖추어야 하는데 한 세션당 한 프로세스를 할당하여 서비스는 방법과 한 쓰레드를 할당하는 방법이 있을 수 있다. 프로세스에 의한 기법은 Context switch, 제어에 대한 부담 등으로 시스템에 사용자수가 증가할수록 시스템 전체적인 부하가 가중되는 반면, 쓰레드에 의한 기법은 자원을 공유하고 신속한 Context switching이 가능하므로 VOD응용에 적합하다. 한편, 이 쓰레드를 사용자 수준에서 스케줄링할 수도 있고 커널에서 지원할 수도 있다. 전자는 커널 자원에 대한 부담은 적지만 한 프로세스에 속한 쓰레드가 블락킹 호출을 하면 소속된 전체 쓰레드가 블락킹된다. 다행히 Linux는 커널 차원에서 쓰레드를 지원하며, 프로세스와 동일한 태스크로 취급한다. 이것은 fork()시스템 콜과는 다른 clone()시스템 콜에 의해서 지원되는데 fork()에 의해 생성된 태스크는 자원을 복사하지만 clone()에 의해 생성된 태스크는 자원을 공유한다. 따라서, SVOD는 Linux의 이 쓰레드로 서버 소프트웨어를 구성한다.

그림 2는 제어 센터와 DB 센터로 구성된 제어 서버이다. 제어 센터는 클라이언트, DB 센터, 그리고 스토리지 서버로 부터의 명령을 일괄적으로 받는 Dispatch thread와 명령을 처리하는 Control thread로 구성되며 이 두 요소 사이는 큐로 연결되어 있다. DB 센터는 각 스토리지 서버에 저장된 비디오 파일에 대한 정보를 유지 및 관리하고, 사용자의 접근에 대한 로그를 기록한다. 제어 센터와 마찬가지로 Dispatch thread는 제어 센터로의 명령을 접수하여, DB thread로 전달한다. DB Thread는 Postmaster 와 접속하여 요구를 처리한다. SVOD는 제어 서버에서 클라이언트나 스토리지 서버의 메시지를 모두 처리하므로, 하나의 세션당 제어 메세시지를 처리하는 구조보다 효율적이며, 스토리지 서버의 부하유지를 효율적으로 조절할 수 있다.

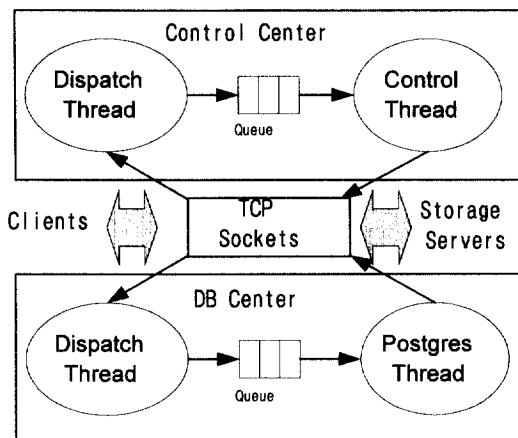



그림 3은 스토리지 서버의 구조를 나타낸다. 스토리지 서버는 DirectShow클라이언트나 제어 서버로부터의 메시지를 TCP/IP 소켓을 통하여 처리하는 Message dispatch thread, 클라이언트의 VCR(Play, Pause, Stop, FF)등의 연산에 대한 허용제어를 수행하고 각 세션을 관리하는 Admission thread, 서버 시스템의 자원을 관리하고 허용제어의 기준을 제시하는 Manager thread, 그리고 하나의 쌍으로 클라이언트에게 데이터를 공급하는 Disk thread, Network Thread로 구성된다. Admission thread는 새로운 세션이 허용되면 Disk thread와 Network thread를 생성하고 자원의 관리를 위해 Session state table에 현재 세션의 상태를 기록한다. Disk thread 는 본 연구에서 프로토타입으로 개발된 Media File System (MFS)를 경유하여 디스크 배열로부터 데이터 블록을 주기적으로 읽어온다. 또한, Network thread는 클

그림 2. Control Server

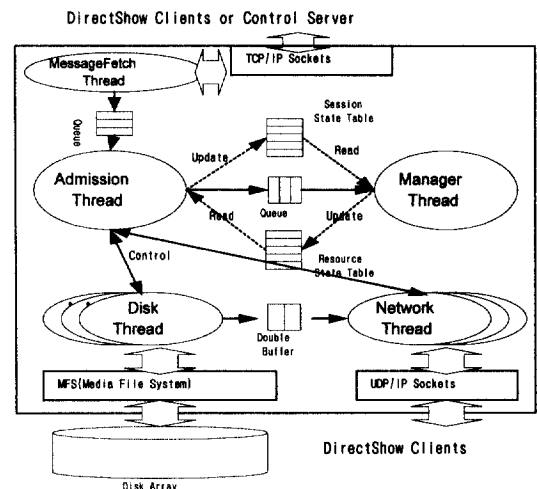



그림 3. Storage Server

라이언트의 VCR연산의 종류에 따라 주기적으로 UDP/IP 소켓으로 데이터를 공급한다. 이 두 쓰레드 사이에는 이중 버퍼로 연결되어서 Disk thread의 데이터 생산과 Network thread의 데이터 소비가 병행된다. 다음 절에서 서버의 데이터 생산과 소비의 스케줄링 모형을 설명한다.

3.3 SVOD의 생산자, 소비자 스케줄링

현재 SVOD는 MPEG-I 파일을 지원한다. MPEG-I은 일반적으로 가변 데이터율(VBR : Variable bit rates)로 압축되어있다. 따라서, 주기마다 디스크나 네트워크의 대역폭 요구량이 변한다. SVOD는 DirectShow COM 인터페이스를 통해 플레이백 될 스트림에 대한 상영시간과 전체 파일의 크기를 고려하여 초기 데이터 요구량을 계산하여 각 세션의 자원의 요구량으로 환산한다. 다음 그림 4, 5는 생산자와 소비자의 스케줄링 모형과 의사 코드이다.

생산자는 스트림의 평균 데이터 요구량을 참조하여 주기 P를 정한다. 생산자는 CPU 처리시간인 ct, 디스크로부터 데이터를 버퍼링하는 시간 bt, 그리고 주기 P에서 ct와 bt를 제한 slack time인 ns로 구성된다. ns는 주기내에 버퍼링을 끝냈을 경우 다른 생산자에게 CPU 제어권을 반납하기 위해서 커널 제공함수인 nanosleep()을 호출한다. 따라서, 생산자는 주기적으로 저장 장치로부터 데이터를 생산하고, 한 주기에서 시간이 남을 경우 CPU를 반납하여 다른 생산자들의 신속한 스케줄링을 지원한다.

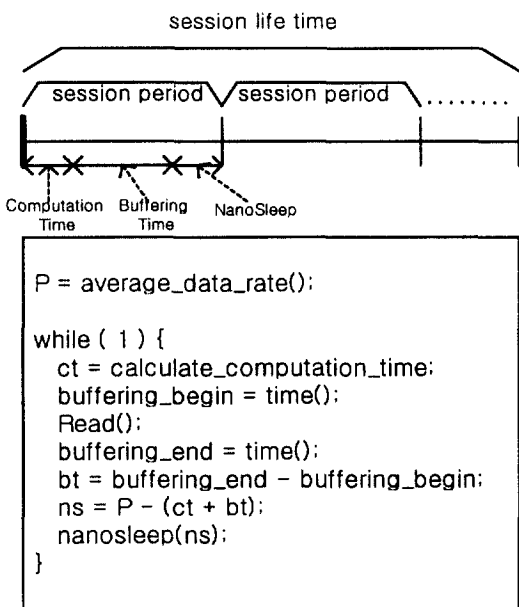
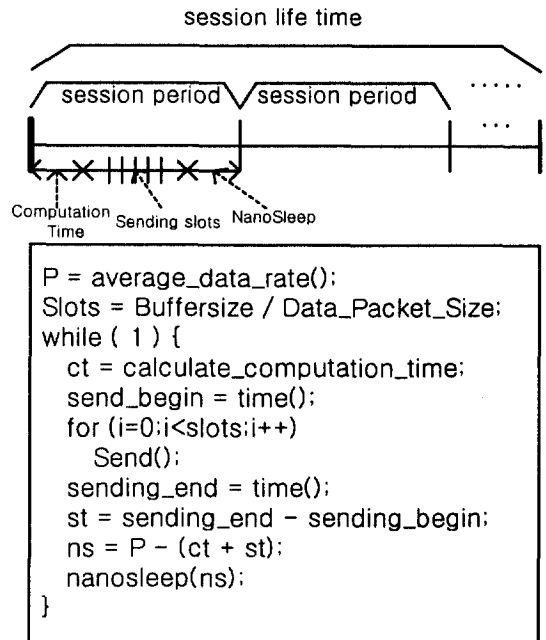



그림 4. 생산자 스케줄링

그림 5. 소비자 스케줄링

소비자도 생산자와 마찬가지로 하나의 버퍼 블록을 전송해야할 주기를 스트림의 평균 데이터 요구량을 참조하여 P를 정한다. 소비자도 CPU 처리시간인 ct, 데이터를 전송하는 시간인 st, 그리고 주기 P에서 ct와 st를 제한 slack time인 ns로 구성된다. 소비자는 한번에 버퍼에 있는 모든 데이터를 전송할 수 없고, UDP가 신뢰성을 보장하지 못하며, 클라이언트가 데이터 패킷을 손실할 수 있으므로 버퍼크기를 한번에 전송할 패킷의 크기로 나눈 nslots을 구하여 이 회수만큼 패킷을 전송하는 방식을 취한다. 소비자인 Network thread도 ns시간 만큼 CPU를 반납하여 시스템 자원을 효율적으로 이용한다. 참고로, SVOD는 비실시간 운영체제인 Linux 커널상에서 구현이 되었기 때문에 Disk, Network thread가 정해진 시간에 스케줄링되어서 주기내에 작업을 마치도록 보장하지는 못한다. 또한, ns값이 음수이면 현재 디스크 자원이나 네트워크의 자원이 충분하지 못함을 의미하며, 이것은 Manager Thread에게 보고되어서 현재 자원의 상태를 갱신한다.

3.4 Media file system (MFS)

기존의 Linux Ext2 파일시스템은 텍스트용으로 일반적인 유닉스 운영체제에 적합하게 설계되어졌다. 그러나, 연속 매체의 특징인 대용량과 고대역폭이라는 특성은 만족시키지 못한다. 또한, 디스크를 관리하는 디바이스 드라이브의 경우에는 디스크의 실시간 스케줄링을 만족시키지 못하고 있다. 따라서, SVOD 시스템은 기존의 Ext2파일 시스템을 개조하여 4킬로 바이트 이상의 블록을 지원하는 MFS와 다수의 디스크를 배열로 묶어서 파일시스템의 논리적 블록을 디스크 배열에 스트라이핑 시켜주는 Media Device Driver(MDD)를 구현하였다. 그림 6은 기존의 커널에 추가된 MFS 및 MDD의 구조를 보여준다. VFS(Virtual file system) 측면에서 MFS는 다른 지역 파일시스템과 마찬가지로 취급된다. 스토리지 서버에서 비디오파일이 저장된 디스크로의 요구(Open, Read, Write etc)가 발생하면 VFS는 자신의 파일시스템 목록에 등록된 MFS의 API로 변환한다. MFS는 기존의 디바이스 드라이브로 파일 엑세스 요구를 보내지 않고 디스크배열을 관리하는 MDD로 보낸다. MDD는 해당 블록을 읽어서 버퍼 캐쉬에 채운다. 현재 구현된 MFS와 MDD는 블록크기와 인기도에 의한 배치 기법만 지원하는 프로토타입 수준이다. 실시간 디스크 스케줄링, 버퍼 캐쉬 전략, 자원 모니터링의 기능이 향후 추가되어야 하며, 현재 실험에서는 인기도에 의한 배치 기법만 대상으로 하고 있다.

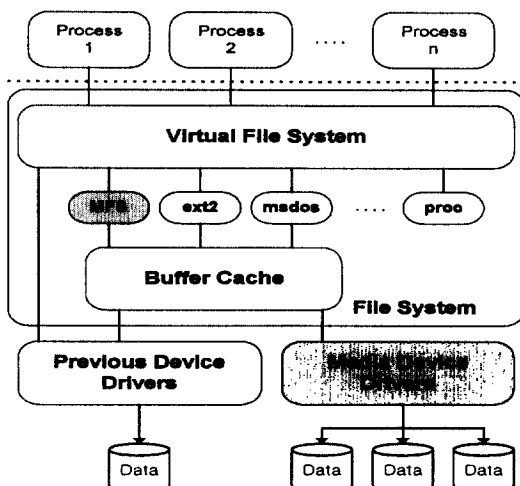



그림 6. MFS의 구조

4. DirectShow를 이용한 클라이언트 구조

SVOD 시스템의 클라이언트는 Microsoft DirectShow®를 이용하여 구현되었다. DirectShow는 Component Object Model(COM) 인터페이스를 사용하여 멀티미디어 객체를 처리하는 윈도우 객체의 일종이다[12]. DirectShow는 대부분의 동영상 포맷(MPEG, AVI, MOV, etc)을 지원하며, 미리 정의된 여러 인터페이스를 통해서 미디어를 제어할 수 있다. 따라서, 사용자들은 값비싼 하드웨어 디코더 없이도 윈도우 운영체제만 설치되어있으면 여러 포맷의 동영상을 재생할 수 있다.

그림 7은 클라이언트의 내부 구조이고 그림 8은 구현된 클라이언트 인터페이스이다. DirectShow를 이용한 클라이언트 프로그램은 크게 세 부분으로 나뉜다. 네트웍 제어를 위한 네트웍 쓰레드, DirectShow의 제어를 위한 VCR 제어 쓰레드, 그리고 메모리 관리 루틴이다. 메시지 & 데이터 쓰레드는 서버와의 통신을 담당하는 쓰레드로서, TCP 소켓으로 제어신호를 처리하고, UDP 소켓으로 실제 동영상 데이터를 수신한다. VCR 제어 쓰레드는 사용자의 입력을 받아서 해당 작업을 DirectShow인터페이스를 사용하여 수행한다. 마지막으로, 메모리 관리모듈은 UDP 소켓으로 수신된 동영상 데이터를 원형 큐를 사용하여 비동기적으로 DirectShow에 공급한다. 이때, 원형 큐의 크기가 제한되어 있으므로 동영상 데이터의 수신속도를 제어하는 부분이 필요하게 된다. 이것은 5장에서 자세히 설명한다. 네트워크용 클라이언트의 구현 결과 DirectShow의 *Source filter*는 32KB의 속도로 샘플링 하는데 클라이언트 버퍼가 32KB를 항상 안정적으로 공급만 할 수 있다면

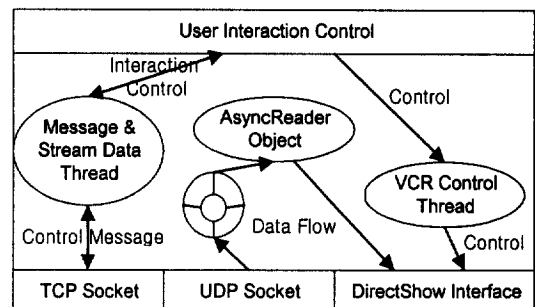



그림 7. 클라이언트의 구조

그림 8. 클라이언트 인터페이스

stand-alone 클라이언트와 동일한 성능을 나타내었다.

5. 클라이언트와 서버 모듈의 프로토콜 및 흐름제어

5.1 클라이언트와 서버 모듈간의 프로토콜

SVOD의 패킷에는 크게 제어신호와 데이터 패킷이 있는데, 제어신호는 클라이언트와 제어센터, 제어센터와 DB센타 그리고, 제어센터와 스트림 관리자로의 신호가 있다. 이 제어신호는 그림 9의 구조체를 패킷화하여 전송한다. 따라서, SVOD내의 모든 동작은 struct Request의 16바이트로 시작되며, TCP 소켓으로 신뢰성있게 전송된다. 그림 9의 구조체에서 "client_id"는 서버에서 클라이언트를 구별하기 위한 식별자로 사용된다. 현재 이 값은 인증과정에서의 사용자의 *id*로 사용된다. "control_id"는 프로토콜 식별자(PLAY, PAUSE, STOP, BUFFERING 등)로서 클라이언트는 서버로 명령의 요구 및 현재 상태를 보고하고, 서버는 자신의 모듈내의 동작을 제어하는데 사용된다. "control_mode"는 "control_id"의 향후

```
struct Request {
        unsigned long   client_id;
        unsigned long   control_id;
        unsigned long   control_mode;
        unsigned long   offset;
};
```

그림 9. 제어 신호 구조체

확장이나 "offset"필드와 조합하여 시스템의 동작을 제어하기 위해서 사용된다. 그리고 "offset"은 "control_id"에 수반되는 구체적인 값으로서 파일의 크기나 흐름제어를 위한 전송률 등을 위한 필드로 사용된다.

그림 10은 클라이언트의 상태도이다. SVOD 클라이언트는 비디오 파일의 첫 블록을 수신하는 즉시 재생할 수 없다. 왜냐하면 초기에 DirectShow의 소비율을 만족시킬 수 없기 때문이다. 또한, DirectShow는 MPEG-1 파일의 헤드정보를 이용해서 필터 그래프를 구성하고 일정량을 버퍼링시키면서 랜더링을 시도하는데 이것이 성공해야 정상적인 플레이백이 가능하다. 랜더링을 만족시키는데 필요한 최소 버퍼량을 다양한 비트율로 압축된 파일들을 대상으로 플레이백해 보았는데 최대 1초 가량 소요됨을 확인하였다. 이것은 MPEG 파일마다 비트율이 가변적이기 때문이다. 따라서, 초기 지연 대기 시간이 1초가 된다. 이 버퍼링이 종료되면 Play할 수 있는 상태로 전환되며, 기타 VCR연산을 그림 10과 같이 수행할 수 있다. SVOD는 모든 VCR연산시 서버로 허용제어 요청을 하는데, 이것은 VCR연산 사이에 서버의 자원 가용도가 변경될 수 있기 때문이다.

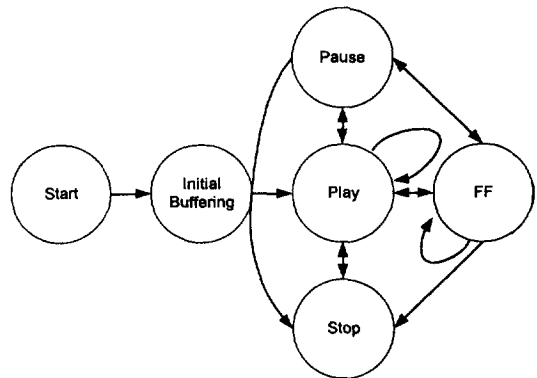



그림 10. 클라이언트 상태도

5.2 흐름제어와 Fast Forward 연산

5.2.1 흐름제어

본 논문에서 서버와 클라이언트간의 데이터 전송은 TCP/IP상에서 UDP패킷을 사용하여 전송한다. UDP는 비연결성 데이터 전송 방법으로서 TCP패킷 방법에 비해 전송률이 높으나, 데이터 전송을 잘 관

리하지 못하면 패킷이 손실되는 약점을 가지고 있다. 또한, VOD서비스를 받는 사용자가 증가함에 따라 서버가 관리하는 세션(Session)이 증가하므로, 세션 간 스위칭에 대한 부하가 증가하게 된다. 그러므로, 데이터를 전송하는 실제 속도에도 영향을 받게 된다. 이에 반해, 클라이언트의 경우 버퍼의 크기에는 제한이 있다. 따라서, 전송속도가 너무 빠르면 UDP패킷의 손실이 발생하게 된다. 반면 사용자의 증가로 클라이언트의 재생율(Playback rates)보다 서버의 실제 전송속도가 느리게 되면 역시 jitter가 발생하게 된다. 따라서, 클라이언트와 서버간에 전송속도를 제어하는 프로토콜과 알고리즘이 필요하게 된다.

SVOD에서는 클라이언트가 버퍼를 주기적으로 모니터링하여 데이터 전송률을 서버에게 요청하는데 그림 10의 Play, FF상태에서 이벤트가 발생된다. 이것은 Leaky Bucket알고리즘[13]의 High water mark와 Low water mark 개념을 이용하여 클라이언트 버퍼의 오버플로우와 언더플로우 발생을 억제한다. 클라이언트가 버퍼의 상태를 모니터링하는 알고리즘은 그림 11과 같다. 초기에 사용자가 선택한 파일의 크기를 DirectShow 인터페이스로 구해진 Playback 시간으로 나누면 초당 전송률이 구해진다. 이 전송률을 서버가 한번에 보낼 패킷의 크기로 나누면 패킷당 전송률이 나오는데 이 값을 서버로 전송한다. 서버에서 이 값은 네트워크 쓰레드의 주기가 된다. MPEG파일은 가변적인 비트율로 부호화되므로

```
Data_rate = (File_size / Playback_time) / Packet_size;
Send Data_rate to Server;
Function Flow_control( )  // 주기적으로 호출되는 Function
Begin
  CQueue_size = Get_queue_size(); // Circular queue size
  Up_down_value = (Current_buffer_size-Previous_buffer_size )
                  / (Current_time   Previous_time)
  If ( CQueue_size > Upper_Bound_of_Safe_State ) {
        Data_rate = Data_rate   Up_down_value;
  Send Data_rate to Server;
    } else if ( CQueue_size < Lower_Bound_of_Safe_State) {
          Data_rate = Data_rate + Up_down_value;
     Send Data_rate to Server;
    }
End
```

그림 11. 흐름제어 알고리즘

클라이언트의 버퍼 소비량도 역시 가변적이 된다. 따라서, 주기적으로 그림 11의 Flow_control() 함수를 호출하여 버퍼를 안정화 시키는데 원형 큐의 유효한 데이터의 크기가 안정화 상태의 상한을 초과하면 현재 전송률에 *Up_down_value*를 감하고, 하한 미만이면 *Up_down_value*를 더한다. 여기서 안정화 상태의 상한과 하한 값의 설정과 *Up_down_value* 의 결정이 VOD시스템의 성능에 상당한 영향을 준다. 상한과 하한의 경우, 두 값의 간격을 좁게 주면 전송속도 조정에 대한 빈번한 제어 메시지 전송으로 서버에 부하가 걸리게 되고, 간격이 너무 넓으면, Overflow되거나 Underflow될 가능성이 높아지게 된다. SVOD에서는 이 값을 경험적으로 구했는데 버퍼 크기의 하한 30%, 상한 70%가 적당함을 알 수 있었다. 또한, *Up_down_value* 는 이전 주기와 현재 주기사이의 데이터 변화량을 측정한 값으로서 실험 결과 현재 버퍼 크기의 ±15% 정도인 것으로 측정되었다. 이 두 경우의 정확한 이론적 근거는 본 연구의 범위를 벗어나므로 향후 연구로 남겨둔다.

5.2.2 Fast Forward (FF) 연산

MPEG-1에서 FF 연산을 지원하기 위해서는 계층화된 압축방법과 부분 전송 등의 기법이 있다. 전자는 FF 연산을 위해 독립적으로 부호화된 파일이 필요하므로 추가적인 저장 공간이 필요하고, 후자의 경우는 저장 서버가 파일의 구조를 분석하여 재 구조화하여 전송해야하므로 CPU의 부담이 증가한다. 따라서, SVOD는 위의 두 방법의 복잡한 구현보다는 단순히 흐름제어를 통해서 FF 연산을 구현하였다. 즉 서버는 FF 연산시 흐름제어를 통하여 단위 시간에 전송하는 데이터의 양을 증가시키고, 클라이언트는 DirectShow ImediaPosition 인터페이스의 put_Rate() 메소드를 이용하여 디코딩 속도를 증가시켰다. 이 메소드는 하나의 파라미터를 입력으로 받는데, 디폴트 1.0 일 때는 정상 플레이백을 하고, 2.0일 때는 두 배의 플레이백을 한다. 이 방법은 디스크나 네트워크의 자원을 비효율적으로 사용하지만 현재 연구의 범위를 벗어나므로 향후 과제로 미룬다.

6. 구현 및 성능 평가

SVOD는 Linux 커널 2.2.2와 gcc 컴파일러을 이용

하여 구현 되었는데, 표 1은 SVOD의 개발 환경 및 소스 코드의 크기를 보여준다. SVOD의 성능을 평가하기 위해서 현재 구축된 서버상에서 실측을 하였는데 표 2는 주요 시스템 변수이고, 표 3은 장착된 디스크의 사양이다.

실험에서는 실제로 플레이백하는 클라이언트를 원하는 수 만큼 구성할 수 없으므로 Unix가 탑재된 시스템에서 시간당 Poisson분포에 의해 다수의 가상 클라이언트를 생성하였다. 생성된 클라이언트는 Zipf

표 1. 개발환경과 소스코드의 크기

구현환경		모듈별 소스코드 크기				
Kernel version	C Compiler		Control Server		Storage Server	Client
			Control center	DB center		
2.2.2	gcc 2.7.2.3	라인수	1,578	781	3,450	32,589
		크기 (바이트)	37,438	17,429	85,394	651,787

표 2. 시스템 변수

Parameters	Value
System	Samsung Smartstation 725
CPU	Pentium II 450MHz
Memory	256 MB
System I/O Bus	Adaptec 2940UW 40MB/sec
Number of Disks	3
Total capacity	13.5GB
Total videos	40
Video types	MPEG-I (about 300MB)
Average playback time	About 30 minutes
Disk block size	32KB
Network packet size	4KB

표 3. 디스크 사양

Parameters	Value
Disk Model	Seagate ST35420W
Formatted Capacity	4.5GB
Recording Method	ZBR (Zone bit recording)
Average Seek Time	9.5 (Read)
Internal Transfer Rate	110 to 193.88 mbits/sec
External Transfer Rate	40 mbyte/sec

[14]분포에 따라서 비디오를 선택하는데, 클라이언트가 접속되면 서버는 동시 사용자수에 대한 deadline miss를 측정하고, 클라이언트는 서비스 품질 저하(service degradation)를 측정한다. 서버의 허용제어 모듈은 부하에 대한 시스템 성능을 측정하기 위해 무조건 새로운 클라이언트를 수용한다. 서버는 단순 Play모드로만 동작하는데 이것은 현재 SVOD가 normal MPEG1 파일만 저장되어 있어, scalable coding으로 저장된 파일을 전송하지 않고 서버의 전송률만 증가하는 것으로 FF를 실험하는 것은 의미가 없기 때문이다. 그리고, 표 1에서 보듯이 서버의 디스크 배열에는 30분용 MPEG-1 (약 300MB) 파일을 40편 저장했는데, 본 연구의 기존 연구인 Linear placement기법[15]에 따라 인기도 순으로 디스크의 외곽 존부터 순차적으로 배치하였다. 장착된 3개의 디스크는 ZBR(Zone bit recording) 디스크로서 MDD에 의해 가상적인 디스크 배열로 구성되며, 하나의 SCSI-2 어댑터에 연결되어 있다.

그림 12는 시간당 평균 100명의 사용자가 Poisson 분포로 서비스를 요청할 때, 분당 동시 사용자수와 분당 deadline miss 발생 회수를 측정한 것이다. 실험의 목적은 클라이언트의 비디오 접근 패턴에 따라 서버의 부하를 측정하는 것이다. 실험에서 deadline miss는 서버의 네트워크 쓰레드가 서비스 주기를 초과한 경우로서, 서버 부하의 정도를 나타낸다. 이 상태가 지속되면 클라이언트의 버퍼가 Underflow되어 서비스의 질이 저하된다. 클라이언트는 그림 10의 상태도에서 처럼, 접속한 후 버퍼링을 통해서 초기의 버퍼를 안정화 시킨 후 Play 모드로 전환되기 때문이다.

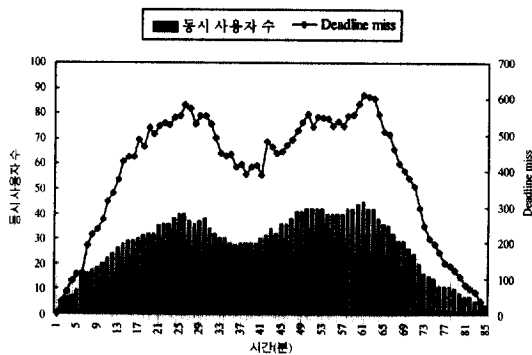
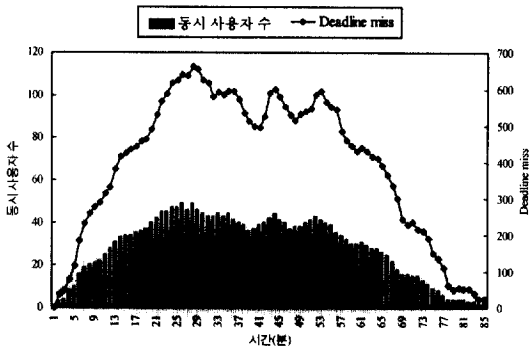
실생활에서 비디오에 대한 선택 확률은 Zipf 분포 값이 0.271 일 경우와 유사하다고 알려져 있는데[14] 그림 12의 (a) 가 그 결과를 나타내고, 비교를 위해서 그림 12의 (b) 는 0.813 일 때의 결과이다. ZBR 디스크에서 비디오 블록의 배치시 인기비디오는 전송률이 높은 외곽 존에 배치하는 것이 서버의 처리율이 증가됨을 이미 확인하였으므로[15], 본 실험에서는 선형 배치시 사용자의 접근 패턴에 대한 동시 사용자의 수와 서버의 부하정도를 측정하였다. 그림 12의 (a)의 경우 서버는 외곽 존에 있는 인기비디오에 대한 선택확률이 증가하므로, 그림 12의 (b)의 0.813 보다 동시 사용자의 수와 deadline miss 발생 회수에 있어서 나은 성능을 보인다. 이것은 0.813일 경우는

(a) (Zipf value : 0.271)

(b) (Zipf value : 0.813)

그림 12. 분당 동시 사용자수에 대한 Deadline miss 회수

(a) (Zipf value : 0.271)

(b) (Zipf value : 0.813)

그림 13. 분당 동시사용자수에 대한 Service degradation 발생 회수

ZBR 전체 디스크의 각 Zone에 대한 접근 확률이 균등분포에 근접하므로 전송률이 낮은 Zone에 대한 서비스 확률의 증가로 인해 서버의 deadline miss 발생 회수가 증가한 것이다.

그림 13은 시간당 200명의 사용자에 대한 클라이언트의 서비스 품질 저하(degradation) 발생 회수를 측정한 것이다. 서버에 동시 사용자가 60명 이하일 때는 그림 13의 (a), (b) 두 경우 모두 클라이언트의 degradation 발생수가 거의 0 에 근접하지만 60명 이상일 경우는 총 클라이언트 수 대비 15% 가량 발생하고, 그림 13의 (b)에서 최대 100 명의 사용자가 서비스 중일 때 최대 20% 정도의 degradation 이 발생하였다. 그림 13의 (b)에 비해서 그림 13의 (a)가 총 degradation 발생 회수에 있어서 나은 성능을 보인 이유는 그림 12에서의 결과와 마찬가지로 전송률이 높은 Zone에 배치한 비디오 블록에 대해서 사용자의 접근 요구가 빈번하게 발생하였기 때문이다. 따라서, 현재 구축된 환경에서의 지원 가능한 동시 사용자의 수는 60명 이하 일 때, 안정적으로 스트림을 전송할 수 있다.

7. 결론 및 향후 연구

본 연구에서는 무료 운영체제인 Linux커널과 저가형 PC 클러스터상에서 확장 가능한 소규모의 VOD시스템을 설계 및 구현하였다. SVOD 시스템의 스트림관리자는 쓰레드 단위로 클라이언트의 세션을 관리하여 서버의 응답성을 증가시켰으며, 다수의 스토리지서버에 다수의 스트림관리자가 탑재되어 서버의 확장성을 지원한다. 또한, VOD에 적합한 MFS(Media File system)와 MDD(Medai device driver)를 설계 및 구현하였으며, 다양한 매체를 플레이백할 수 있는 소프트웨어 디코더인 DirectShow 클라이언트를 구현하였다.

향후 연구로는 MFS, MDD의 실시간성의 지원, 시스템 전체의 QoS(Quality of Service)의 보장, 클라이언트 버퍼의 안정화를 위한 이론적인 근거, 그리고, 네트워크 자원의 예약을 위한 RSVP(Resource Reservation Protocol)의 적용 등이다.

참 고 문 헌

[1] http://www.seagate.com/disc/medalist/mproscsi.shtml

[2] http://www.adaptec.com/products/overview/scsi3950u2.html

[3] M. M. Buddhikot, X. J. Chem, D. Wu, and G. M. Parukar, "Enhancements to 4.4 BSD UNI for Efficient Networked Multimedia in Project MARS", IEEE Multimedia systems, pp.316-325, 1998.

[4] S. R. Tong, Y. F. Huang, and J. C. L. Liu, "Study on Disk Zoning for Video Servers", IEEE Multimedia systems, pp.86-95, 1998.

[5] J. W. Kim, Y. L. Lho, and K. D. Chung, "An Effective Video Block Placement Scheme on VOD Server based on Multi-Zone Recording Disks", IEEE Multimedia Systems, pp.29-36, 1997.

[6] C. Martin, P. S. Narayan, B. Ozden, R. Rastogi and A.Silberschatz, "The Fellini Multimedia Storage Server," Multimedia Information Storage and Manangement, Edited by S.M. Chung, Kluwer Academic Publishers, pp.117-146, 1996.

[7] S. Grandeharizad도, R. Zimmermann. W. Shi. R. Rejaie, D. Ieraridi, and T. W. Li, "Mitra: A Scalable Continuous Media Server," Kluwer Multimedia Tools and Applications, 5(1), pp.79-108, 1997.

[8] J. Gafsi, U. Walther, and E. W. Biersack, "Design and Implementation of a Scalable, Reliable, and Distributed VOD-Server," Proc. of the 5th joint IFIP-TC6 and ICCC Conference on Computer Communications, 1998.

[9] R. L. Haskin and F. L. Stein, "A system for the delivery of interactive television programming", in Compcon: Digest of papers, pp.209-215, 1995.

[10] F. A. Tobagi and J. Pang, "Starworks a video applications server", Proceedings of Compcon, (San Fransisco, CA), February 22-25 1993.

[11] M Beck, H bohme, M Dziadzka, U kunitz, R Magnus, D Vervorner, "Linux Kernel Internals", pp.55-58, Addison wesley.

[12] http://www.microsoft.com/directx/resources/dx6mediasdk.htm

[13] Andrew S.Tanenbaum, "Computer Networks", pp.380-381, Prentince Hall, 1996.

[14] Zipf, G., Human Behavior and the Principle of Least Effort, Addison Wesley, 1949.

[15] J. Kim, Y. Lho, K. Chung., "An Effective Video Block Placement Scheme on VOD Server based on Multi- Zone Recording Disks," Proceedings of the IEEE International conference on Multimedia Computing and Systems, Ottawa, 1997.

김 정 원

1995년 2월 부산대학교 전자계산학과 졸업(이학사)
1997년 2월 부산대학교 전자계산학과 대학원 졸업(이학석사)
1999년 2월 부산대학교 전자계산학과 박사 수료
관심분야 : 멀티미디어, VOD, 멀티미디어 네트워크

김 인 환

1997년 2월 부산대학교 전자계산학과 졸업(이학사)
1999년 2월 부산대학교 전자계산학과 대학원 졸업(이학석사)
1999년~현재 부산대학교 전자계산학과 박사과정
관심분야 : 멀티미디어, VOD, 실시간 운영체제

정 기 동

1973년 서울대학교 공과대학 공학사
1975년 서울대학교 대학원 석사
1986년 서울대학교 대학원 계산통계학과 박사
1990~1991년 MIT, South Carolina 대학 교환교수
1978~현재 부산대학교 정보·컴퓨터공학부 교수
관심분야 : 멀티미디어, 병렬처리, 운영체제